kivitendo Entwickeln

Werkzeuge und Konzepte

G. Richardson, kivitec GmbH, Bonn 2017

Übersicht

- Konsole konfigurieren
- psql konfigurieren
- Editor konfigurieren
- Debugging
- kivitendo console
- SL::Dev::ALL
- Tests

Konsole konfigurieren

Aliase

.zshrc

```
alias cdg="/usr/local/src/kivitendo/"

if [[ $HOST == "mars" ]]; then
  hash -d git='/usr/local/src/kivitendo/'
  hash -d sql='/usr/local/src/kivitendo/sql/Pg-upgrade2'
  hash -d temp='/usr/local/src/kivitendo/templates/webpages'
fi

alias taglist="cdg; perl $HOME/bin/pltags.pl **/*.pl **/*.pm **/*t";
```

Aliase

.zshrc

```
lxrose() {
  ROSEPARAM='--all'
  if [[ -z "$1" ]]; then
    ROSEPARAM=$1
  fi
  cdg
  perl scripts/rose_auto_create_model.pl --client testdb $ROSE
  cd -
}
```

psql Umgebungsvariablen

.zshrc

```
export PSQL_EDITOR='vim +"set syntax=sql" '
export PSQL_EDITOR_LINENUMBER_ARG='+'
export PAGER=less
```

git Aliase

.zshrc

```
alias gc="git commit"
alias gd="git diff"
alias gg="git grep"
alias ga="git add"
alias gau="git add -u"  # add all modified and deleted files,
alias gap="git add --patch"
alias gcb="git checkout -b"
alias gco="git checkout"
alias gcp="git cherry-pick"
```

git Aliase

.zshrc

```
alias gits="git status -uno"
alias gdc="git diff --cached"
alias go="git ls-files --others"
alias gcm="git commit -m"

alias glg="git log --graph --oneline --decorate --all" # git log grap
alias gll="git log --stat --oneline" # show which lines changed in co
alias glo="git log --oneline" # show short version

# order git branches by date
alias gbh="git for-each-ref --sort=-committerdate refs/heads/"
```

git branch in Prompt

https://pilif.github.io/2008/04/git-branch-in-zsh-prompt/"

```
% gco master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 5 commits.
  (use "git push" to publish your local commits)
geoff@servername ~uls/kivitendo-geoff (master)

% gco fibuschnellsuche
Switched to branch 'fibuschnellsuche'
geoff@servername ~uls/kivitendo-geoff (fibuschnellsuche)
```

zsh autocompletion

https://github.com/zsh-users/zsh-completions/blob/master/src/_pgsql_utils

```
apt-get install libpq-dev
```

.zshrc

```
fpath=($HOME/.zsh-completions $fpath)
zmodload zsh/complist
autoload -U compinit && compinit
```

```
 % psql test[TAB]
- PostgreSQL database -
test  test30  testefeste    testupgrade   test-upgrade
```

psql Konfigurieren

mit Datenbank verbinden

```
$ psql --help | grep -A 3 Connection
Connection options:
  -h, --host=HOSTNAME      database server host or socket directory (
  -p, --port=PORT          database server port (default: "5432")
  -U, --username=USERNAME  database user name (default: "geoff")

$ psql -h 127.0.0.1 -p 5432 -U geoff
```

.pgpass

Passwörter speichern

```
localhost:5432:*:postgres:MYSECRET
```

.pg_service.conf

```
[test]
host=192.168.5.10
port=5433
dbname=kivitendo
user=geoff
password=foobar

[prod]
host=192.168.5.20
port=5432
dbname=kivitendo
user=geoff
password=foobar
```

```
export PGSERVICE=prod && psql
```

.psqlrc

```
\conninfo
\pset null ☠
\x auto
\set COMP_KEYWORD_CASE upper
\timing
\pset pager always
\set PROMPT1 '%M:%> %n@%/%R%#%x '
```

psql

```
% psql g_testdb
You are connected to database "g_testdb" as user "geoff" \
  via socket in "/var/run/postgresql" at port "5432".
Null display is "☠".
Expanded display is used automatically.
Timing is on.
Pager is always used.
type :version to see version

Time: 0,102 ms
psql (9.3.14)

[local]:5432 geoff@g_testdb=>
```

.psqlrc

```
\echo 'type :version to see version\n'
\set version 'SELECT version();'

\set clear '\\! clear;'

\set index_hit_rate 'SELECT relname,
                     100 * idx_scan / (seq_scan + idx_scan) percent_of_times_index_used,
                     n_live_tup rows_in_table
                     FROM pg_stat_user_tables
                     WHERE seq_scan + idx_scan > 0 ORDER BY n_live_tup DESC;'
```

psql

Letzten Befehl bearbeiten

```
g_testdb=> select partnumber from parts limit 3;
           partnumber
--------------------------------
 StL123-______-______-______-Sx
 StL123-______-______-______-TL
 StL123-______-______-______-T_
(3 rows)

g_testdb=> \e
```

psql

Pager wechseln

```
\setenv PAGER 'vim -R -'
```

.psqlrc

```
\set vim '\\setenv PAGER \'vim -R -\''
\set less '\\setenv PAGER less'
```

```
:vim
:less
```

psql Nützliche Befehle

SQL-Datei einlesen:

```
\i sql/Pg-upgrade2/debug.sql
```

Ausgabe in Datei umlenken

```
\o /tmp/out.sql
select partnumber from parts;
\o
```

History anzeigen

```
\s
```

psql Variablen

```
psql -e g_testdb
```

```
\set fromdate 2016-01-01
\set todate 2016-12-31
select sum(amount) from ar where transdate >= :'fromdate'
                             and transdate <= :'todate';

select sum(amount) from ar where transdate >= '2016-01-01'
                             and transdate <= '2016-12-31';
 sum
-----
   ☠
```

psql Variablen

Variablen beim Start von psql setzen:

```
psql -v fromdate=2012-01-01 -v todate=2012-03-31 g_testdb
```

```
\echo :todate
-- 2012-03-31
```

Editor konfigurieren

am Beispiel von vim

Defaults

```
syn on
set sw=2
set ts=2
set et
set ai
set ic
set ruler
set backspace=2
set ml
```

Style guide:

https://www.kivitendo.de/doc/html/ch04s06.html

Trailing whitespace

```
:%s/\s+$//
```

```
highlight ExtraWhitespace ctermbg=red guibg=red
match ExtraWhitespace /\s\+$/
```

```
autocmd BufWritePre *pl %s/\s\+$//e
autocmd BufWritePre *pm %s/\s\+$//e
```

.vimrc

```
" autosource vimrc.lxoffice
autocmd BufNewFile,BufRead  bin/mozilla/*pl :source ~/.vim/vimrc.lxof
autocmd BufNewFile,BufRead  scripts/*pl :source ~/.vim/vimrc.lxoffice
autocmd BufNewFile,BufRead  SL/*pm :source ~/.vim/vimrc.lxoffice
autocmd BufNewFile,BufRead  *.t :source ~/.vim/vimrc.lxoffice
autocmd BufNewFile,BufRead  *.t set filetype=perl
autocmd BufNewFile,BufRead  *.console filetype=perl
autocmd BufNewFile,BufRead  *.console ~/.vim/vimrc.lxoffice
```

snippets

```
snippet form
        $::form->{"${1:VAR}"} ${2}

snippet find_by
 SL::DB::Manager::${1:class}->find_by( ${2:var} => '${3:var}' )

snippet get_all
 SL::DB::Manager::${1:class_name}->get_all( ${2:var} )
```

kivitendo Modules

```
alias perlwc="perl -wc -MSL::Dispatcher"
```

Ab Version 3.5:

```
alias T='perl -Imodules/fallback -Imodules/override -I.'
```

make

.vimrc

```
set autowrite       " automatically save file when make is call
set makeprg=~/bin/efm_perl_kivitendo.pl\ -c\ %\ $*
```

vim

```
:make
"SL/Dev/Inventory.pm" 266L, 9137C written
:!/home/geoff/bin/efm_perl_kivitendo.pl -c SL/Dev/Inventory.pm
SL/Dev/Inventory.pm:25:"my" variable $wh masks earlier declara
```

Fehler landen in Quickfix window

git Integration

fugitive Plugin

- Ggrep
- Gblame
- Gdiff
- Gstatus

```
" run Ggrep for word under cursor
nnoremap gG :Ggrep <cword><CR>
" automatically open the quickfix window when Ggrep is called:
autocmd QuickFixCmdPost *grep* cwindow
```

Debugging

Six stages of debugging

1. That can’t happen.
2. That doesn’t happen on my machine.
3. That shouldn’t happen.
4. Why does that happen?
5. Oh, I see.
6. How did that ever work?

Debugging

Print in Datei

```
use Data::Dumper;

open(FH, ">/tmp/test.txt") or die "Can't open file\n";
print FH $form->{partnumber} . "\n";
print FH Dumper($form);
close FH;
```

```
snippet openfh
        open(FH, ">/tmp/${1:FILENAME}") or die "Can't open fil
        print FH "${2:PRINT}";
        close FH;
```

lxdebug

kivitendo.conf

```
[debug]
global_level = DEBUG1
file_name = /tmp/kivi-debug.log
```

Im Code:

```
$main::lxdebug->message(0, "entered loop");
$main::lxdebug->message(0, "invnumber: " . $::form->{invnumber});
$main::lxdebug->dump(0, "form", $::form );
```

Terminal:

```
tail -f /tmp/kivi-debug.log
```

snippets

```
snippet lxdebug
	$main::lxdebug->message(0, "${1:TEXT}");

snippet lxdump
	$main::lxdebug->dump(0, "${1:TEXT}", ${2:ref} );

snippet lxdumpform
	$main::lxdebug->dump(0, "form", $::form );

snippet watchdog
	$::form->{"Watchdog::${1:VAR}"} = 1;
```

lxdebug

Im Template

```
[%- USE L -%]
[% # L.dump(SELF) %]
```

In Tests

```
print "invnumber = $invnumber\n";
diag("invnumber = $invnumber");
```

```
sub dbg {
  # diag(@_);    # wird immer ausgedruckt
  # print(@_);
  # $main::lxdebug->message(0, @_);
}
```

Im Testcode:

```
dbg("Database exists; dropping");
```

kivitendo console

kivitendo console

```
perl scripts/console --client 'testdb' --login unittest
```

```
5+5
# 10

$::form->format_amount(\%::myconfig, 12345.789, 2);
# 12.345,79

my $customer = SL::DB::Manager::Customer->get_first;
$customer->displayable_name
# 23 Testkunde

printf("%s %s\n", $_->customernumber, $_->name) foreach @{ SL::DB::Manager::Customer->get_all() };
# 1 Testkunde
# 2 Testkunde
```

kivitendo.conf

```
[console]
# autologin to use if none is given
client = Unit-Tests
login = unittests

# autorun lines will be executed after autologin.
autorun = require "bin/mozilla/common.pl";
        = use English qw(-no_match_vars);
        = use List::Util qw(min max sum first);
        = use SL::DB::Helper::ALL;
        = use SL::Dev::ALL;

# location of history file for permanent history
history_file = users/console_history
```

Datei ausführen

```
perl scripts/console -f file.console
```

Datei in vim ausführen

```
:!perl scripts/console -f %
```

Automatisch beim Speichern ausführen

```
:autocmd BufWritePost file.console execute \
  '! perl scripts/console -f file.console'
```

SL::Dev::ALL

SL::Dev::ALL

- Schnell Daten für Tests generieren
- Sinnvolle Defaults
- Viele Beispiele im POD

```
% ls -1 SL/Dev
ALL.pm
CustomerVendor.pm
Inventory.pm
Part.pm
Record.pm
```

https://github.com/kivitendo/kivitendo-erp/blob/master/SL/Dev/

```
perldoc SL::Dev::Part
```

SL::Dev Beispiele

Kunde anlegen

Ziel: sinnvolle Defaults für Pflichtfelder (z.B. Steuerzone, Währung, ...)

```
my $cusmin = SL::Dev::CustomerVendor::create_customer->save();
```

```
my $customer = SL::Dev::CustomerVendor::create_customer(
  name         => 'Monsters Inc',
  hourly_rate => 50,
  taxzone_id  => 2,
)->save;

$customer->displayable_name
# 2 Monsters Inc
$customer->currency->name
# EUR
SL::DB::Default->get->currency->name
# EUR
```

Artikel erstellen und einlagern

```
my ($wh, $bin) = SL::Dev::Inventory::create_warehouse_and_bins();
my $part       = SL::Dev::Part::create_part->save;

SL::Dev::Inventory::set_stock(
  part    => $part,
  bin_id => $wh->bins->[2]->id,
  qty     => 5
);

SL::Dev::Inventory::transfer_stock(
  part     => $part,
  from_bin => $wh->bins->[2],
  to_bin   => $wh->bins->[4],
  qty      => 3
);
```

100 Verkaufsaufträge anlegen

```
SL::Dev::Record::create_sales_order(
  save         => 1,
  taxincluded  => 0,
  orderitems => [ SL::Dev::Record::create_order_item(
                    part      => SL::Dev::Part::create_part->save(),
                    qty       => int(rand(12))+1,
                    sellprice => 9 )
                ]
) for 1 .. 100;
```

```
SL::DB::Manager::Order->get_all_count();    # 100
SL::DB::Manager::Part->get_all_count();     # 100
my $qty = sum map { $_->qty } @{ SL::DB::Manager::OrderItem->get_all };  # 1379
sql('select sum(qty) from orderitems');  # 1379
SL::DB::Manager::Order->get_first->amount_as_number  # 74,97
```

Workflow abbilden

```
my $sales_order = SL::Dev::Record::create_sales_order(
  save       => 1,
  orderitems => [ SL::Dev::Record::create_order_item(part => $part1, qty => 11),
                  SL::Dev::Record::create_order_item(part => $part2, qty => 12),
                  SL::Dev::Record::create_order_item(part => $part3, qty => 13),
                ]
);
my $sales_delivery_order = $sales_order->convert_to_delivery_order;
my $invoice              = $sales_delivery_order->convert_to_invoice();
$invoice->pay_invoice(chart_id      => $bank->chart_id,
                      transdate     => DateTime->now->to_kivitendo,
                      memo          => 'foobar',
                      source        => 'barfoo',
                      payment_type  => 'with_skonto',
                     );
```

Tests

Verschiedene Arten von Tests

- Unit Tests
- Integration Test
- Functional Test

Nicht alle Tests brauchen Daten aus der Datenbank

minimaler Test

```
use strict;
use Test::More;

use lib 't';
use Support::TestSetup;
Support::TestSetup::login();

is(1, 1, "1 == 1 ok");

clear_up();
done_testing;
```

Die kivitendo-Test-Suite

https://www.kivitendo.de/doc/html/ch04s05.html

kivitendo Testsuite

```
% perl t/test.pl --fast
t/000setup_database.t .. ok
t/002goodperl.t .................................ok
t/004template.t .................................ok
...
t/template_syntax.t .............................ok
t/wh/transfer.t .................................ok
All tests successful.

Test Summary Report
-------------------
t/db_helper/record_links.t   (Wstat: 0 Tests: 66 Failed: 0)
  TODO passed:   35
  Files=64, Tests=32142, 146 wallclock secs
  Result: PASS
```

Einzelnen Test ausführen

Datenbank neu initialisieren:

```
% perl t/test.pl t/000setup_database.t
 t/000setup_database.t .. ok
 All tests successful.
 Files=1, Tests=1, 78 wallclock secs
 Result: PASS
```

format_amount

```
$config->{numberformat} = '1.000,00';

is($::form->format_amount($config, '1e1', 2), '10,00', 'format 1e1 (numberformat: 1.000,00)');
is($::form->format_amount($config, 1000, 2), '1.000,00', 'format 1000 (numberformat: 1.000,00)');
is($::form->format_amount($config, 1000.1234, 2), '1.000,12', 'format 1000.1234 (numberformat: 1.000
is($::form->format_amount($config, 1000000000.1234, 2), '1.000.000.000,12', 'format 1000000000.1234
is($::form->format_amount($config, -1000000000.1234, 2), '-1.000.000.000,12', 'format -1000000000.12
```

t/form/format_amount.t

Mock objects

Kein Zugriff auf Datenbank

SL::DB::Part

```
sub displayable_name {
  join ' ', grep $_, map $_[0]->$_, qw(partnumber description);
}
```

```
my $part = SL::DB::Part->new(
  partnumber  => '12',
  description => 'Sugar'
);
$part->displayable_name
# 12 Sugar
```

Einzelnen Test ausführen

```
% perl -Imodules/fallback -Imodules/override -I. t/bank/bank_transactior
1..100
ok 1 - currency_id has been saved
ok 2 - ar amount has been converted
ok 3 - ar amount has been converted
ok 4 - ar transaction doesn't have taxincluded
ok 5 - 8400: has been converted for currency
...
ok 98 - test_ap_payment_p_transaction: bt amount ok
ok 99 - test_ap_payment_p_transaction: paid ok
ok 100 - test_ap_payment_p_transaction: bt invoice amount for ap was ass
```

Test aus vim aufrufen

```
vim t/bank/bank_transaction.t
```

```
:execute "autocmd BufWritePost " . @% . " execute '!perl -MSL::Dispatcher " . @% . \
  "'" | echo @% . " now has a autocmd BufWritePost, delete with :autocmd! BufWritePost"
```

```
function! Lxbufwrite()
execute "autocmd BufWritePost " . @% . " execute '!perl -MSL::Dispatcher " \
  . @% . "'" | echo @% . " now has a autocmd BufWritePost, delete with :autocmd!
endfunction
```

```
:call Lxbufwrite()
```

vim autocommand

```
vim SL/Controller/BankTransaction.pm
```

```
:autocmd BufWritePost SL/Controller/BankTransaction.pm execute \
  '!perl -MSL::Dispatcher t/bank/bank_transaction.t'
```

Testdaten aufräumen

Jeder Test sollte seine erstellten Daten wieder aufräumen

```
sub clear_up {
  SL::DB::Manager::BankTransaction->delete_all(all => 1);
  SL::DB::Manager::InvoiceItem->delete_all(all => 1);
  SL::DB::Manager::InvoiceItem->delete_all(all => 1);
  SL::DB::Manager::Invoice->delete_all(all => 1);
};

clear_up();
done_testing();
```

```
foreach (qw(AccTransaction Invoice Customer)) {
    "SL::DB::Manager::${_}"->delete_all(all => 1);
}
```

Testdaten nutzen

Kommentiert man das finale clear_up() aus, oder bricht den Test nach der initialen Datenerstellung ab, kann man sich an der Oberfläche anmelden und die Tests per Hand ausführen

Beispiel: Kontoauszug verbuchen

t/bank/bank_transactions.t

Test::More

```
is ($part_unit->get_stock, '1000.00000',
     "stock of part_unit in kg (base_unit)");

ok($do1->donumber eq "L20199",
     'Delivery Order Number created');

is_deeply $csv->get_data, [ { description => 'Kaffee' } ],
     'simple case works';
```

perldoc Test::More

Sessions beenden

```
git stash
git checkout kundenbranch
perl t/test.pl t/000setup_database.t
# is in use
```

Ab Postgres Version 9:

```
SELECT pg_terminate_backend(pg_stat_activity.pid)
FROM pg_stat_activity
WHERE pg_stat_activity.datname = 'g_testdb';
```

```
\set terminate_testdb 'SELECT pg_terminate_backend(pg_stat_activity.pid)
                       FROM pg_stat_activity
                       WHERE pg_stat_activity.datname = \'g_testdb\';'
```

vim Plugins

Align

```
my %part_defaults = (
  sellprice => $default_sellprice,
  buchungsgruppen_id => $buchungsgruppe->id,
  unit => $unit->name,
  warehouse => $wh,
  bin => $bin1,
);
```

```
:Align =>
```

Align

```
psql -e g_testdb
$ \a \t \x
Output format is unaligned.
Showing only tuples.
Expanded display is off.
$ select * from units limit 2;
Stck|☠|☠|dimension|1|1
psch|☠|0.00000|service|2|2
:vim
$ select * from units;
:Align |
```

git im Editor

vim Plugin: Fugitive

- Ggrep
- Gblame
- Gdiff
- Gstatus

Formfelder füllen

per URL:

controller.pl?
action=OrderItem/search&customer_id=909&part=foo

```
[% L.customer_vendor_picker('filter.order.customer.id', FORM.customer_id, type='customer') %]
[% L.input_tag('filter.part.all:substr:multi::ilike', FORM.part) %]
```

Formfelder füllen

per jquery in javascript console

```
$("#auth_login").val("kivitendo");
$("#_AUTH_password").val("secret");
$("#_AUTH_client_id").val("5");
$('form').submit();
```